I2C Protocol Implementation

Using the Verilog HDL

Project Report

Amit Bhorania
Ashutosh Gajankush
Ajinkya Bobade
Ravi Patel
Vaibhav Jindal

Stevens Institute of Technology

Dept. of Electrical & Computer Engineering

Digital & Computer System Architecture (CPE-517A)

CERTIFICATE

This is to certify that the project report entitled “I2C PROTOCOL IMPLEMENTATION” done by Amit Bhorania, Ashutosh Gajankush, Ajinkya Bobade, Ravi Patel and Vaibhav Jindal is an authentic work carried out by them at Stevens Institute of Technology, Hoboken, NJ, USA under my guidance. The matter embodied in this project work has not been submitted earlier for the award of any other degree to the best of my knowledge and belief.

ACKNOWLEDGEMENT

We would like to articulate our profound gratitude and indebtedness to our project guide Prof. Narayan Ganesan who has always been a constant motivation and guiding factor throughout the project time in and out as well. It has been a great pleasure for us to get an opportunity to work under him and complete the project successfully.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

1.1 Project Definition

The goal of the project is to design and implement the I2C (Inter-Integrated Circuit) Slave hardware module. The design was described using the Verilog Hardware Description Language using the Xilinx ISE Design Suite.

1.2 Importance

Usage of sensors are increasing in the day to day life and in the industry. Sensors needs some interface to send out the sensed data to the other device. Size of the processor & chips are decreasing with the use of latest technology. Therefore it is preferable to have less number of pins on the chip.

To send the sensor data, two choices are available: Parallel Communication and Serial Communication. Parallel communication provides the higher speed but requires more number of pins for data transfer. On the other hand, Serial communication requires less number of pins and support lower speed. As sensor data transfer requires lower speed, serial communication is the best choice for the sensor chips. Examples of serial communication protocols are UART, I2C, SPI, PCI Express etc.

Among the serial communication protocols, I2C is very useful and famous protocol due to its features. I2C is a two wire bi-directional serial protocol. It uses SCL (Serial Clock Line) and SDA (Serial Data Line) for data transfer. I2C uses the Master Slave approach for communication.

Due to the ease of usage and its features, we decided to implement the I2C slave module as the communication interface for the sensor chips. Sensor chip vendors can directly incorporate this developed I2C slave module in their chip and it can reduce the time to design the chip and reduce the cost.

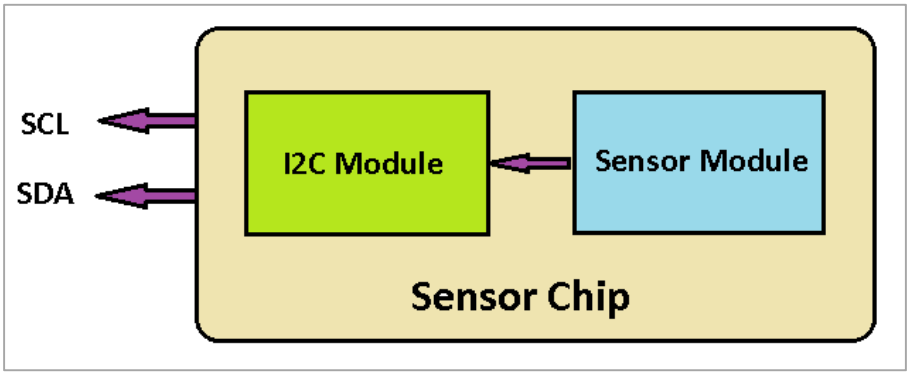

Figure 1.1 Sensor Chip Design

2 I2C Interface Specification

The Interconnect Integrated Circuit or I^2C interface was originally developed by Philips Semiconductors Company for data transfer among ICs at the Printed Circuit Board (PCB) level in early 1980s. This company became NXP Semiconductors which are maintaining the I2C bus specification. All I^2C-bus compatible devices have an interface allowing them to communicate directly with each other via the I^2C-bus. The concept provides an excellent solution for problems in many interfacing in digital design. It is used to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. I^2C is now broadly adopted by many leading chip design companies like Intel, Texas Instrument, Analog Devices, *etc*.

I^2C has two bi-directional lines to carry information between devices. One is the Serial Data (SDA) line and another is Serial Clock (SCL) line. The devices that are connected can be recognized by a unique 7 or 10 bits address. Master, is the one who initiates the communication all the time. All the other devices on the bus are considered Slaves. Normally, Masters are Microcontrollers. The I^2C bus is a multi-Master bus, but only one Master can initiate a data transfer at any time.

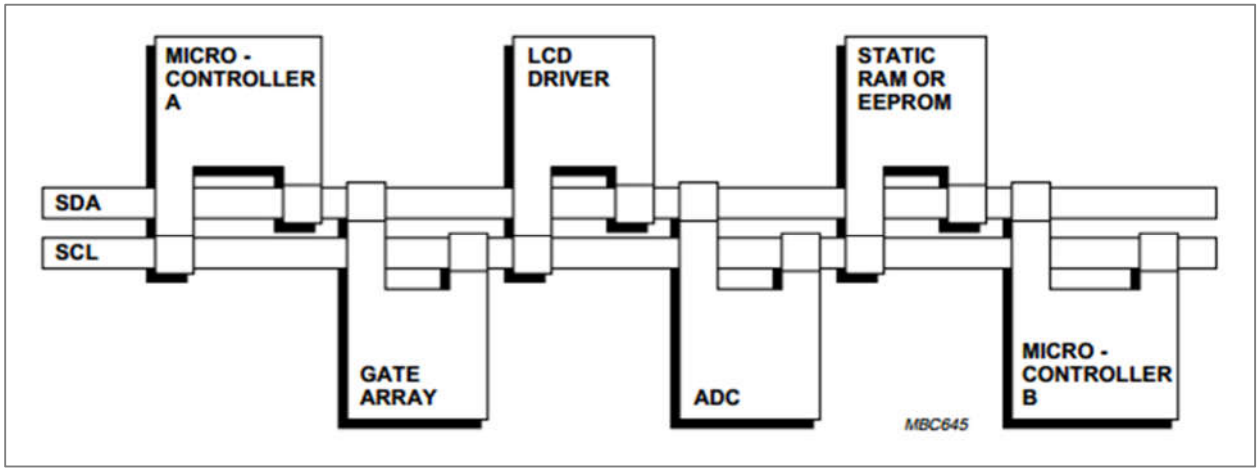

Figure 2.1 I2C Configuration

In the figure above, there is one Master; the other devices are all Slaves. When the master (your controller) wishes to talk to a slave (our CMPS03 for example) it begins by issuing a start sequence on the I2C bus. A start sequence is one of two special sequences defined for the I2C bus, the other being the stop sequence. The start sequence and stop sequence are special in that these are the only places where the SDA (data line) is allowed to change while the SCL (clock line) is high. When data is being transferred, SDA must remain stable and not change whilst SCL is high. The start and stop sequences mark the beginning and end of a transaction with the slave device.

Further details of the signals used in the I2C protocol are described in the following sections.

2.1 Physical Connection on the Bus

Both the SDA line and the SCL line are bi-directional. Therefore, for a particular device, the bus can be controlled by the master or by one of many slaves. For this to be implemented, a wired-OR function is employed.

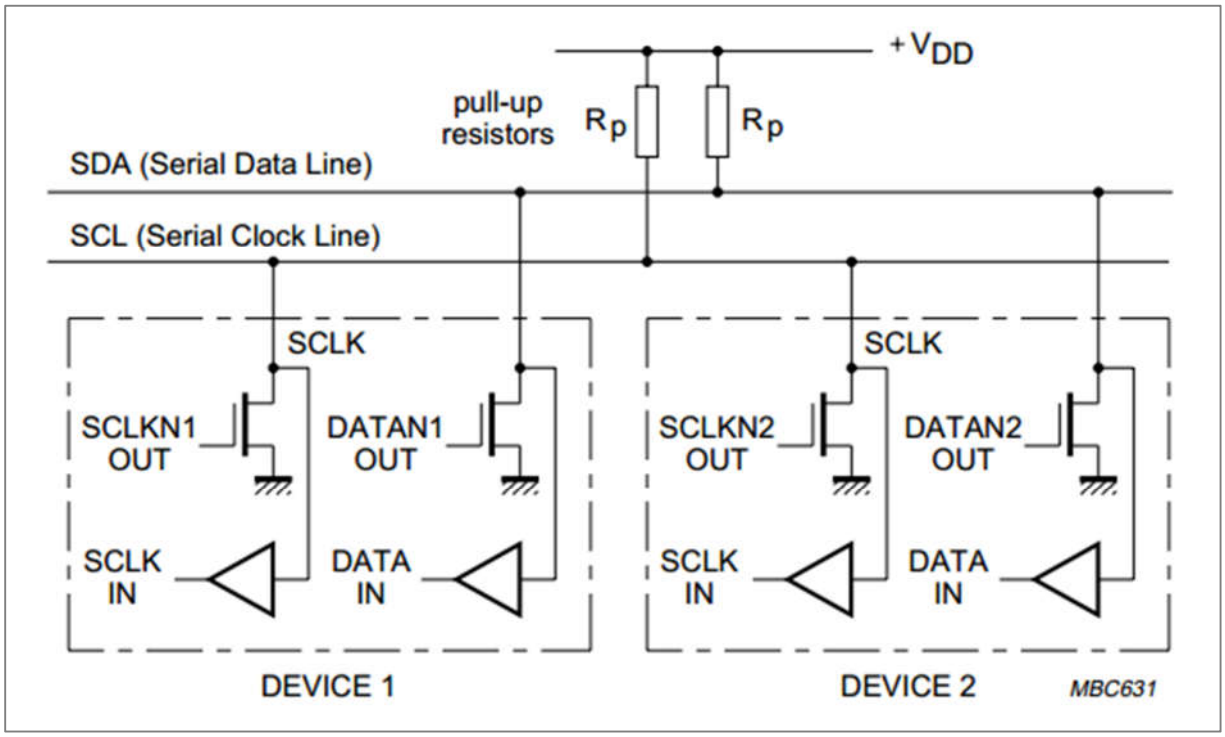

Figure 2.2 I2C Physical Connection

To perform the wired-OR function, the outputs of the devices must be connected to the bus using an open-collector bi-polar (or open-drain field-effect) transistor. The bus is connected to the supply voltage (V_{dd}) using a pull-up resistor, $R_{pull\text{-}up}$. The data transfer rate can reach up to 100 KBit/sec in the Standard-mode, up to 400 KBit/sec in the Fast-mode and up to 3.4 Mbits/s in the High-speed mode. When the bus is free, both lines are high. Whenever a device wants to control the bus, it need to monitor the bus for some amount of time. If the bus is free, the device can put a signal on the bus by driving the output transistor, pulling the bus to a “Low” level.

2.2 Start and Stop Conditions

To initiate the address frame, the master device leaves SCL high and pulls SDA low. This puts all slave devices on notice that a transmission is about to start. If two master devices wish to take ownership of the bus at one time, whichever device pulls SDA low first wins the race and gains control of the bus. It is possible to issue repeated starts, initiating a new communication sequence without relinquishing control of the bus to other masters; we'll talk about that later.

Once all the data frames have been sent, the master will generate a stop condition. Stop conditions are defined by a 0->1 (low to high) transition on SDA *after* a 0->1 transition on SCL, with SCL remaining high. During normal data writing operation, the value on SDA should **not** change when SCL is high, to avoid false stop conditions.

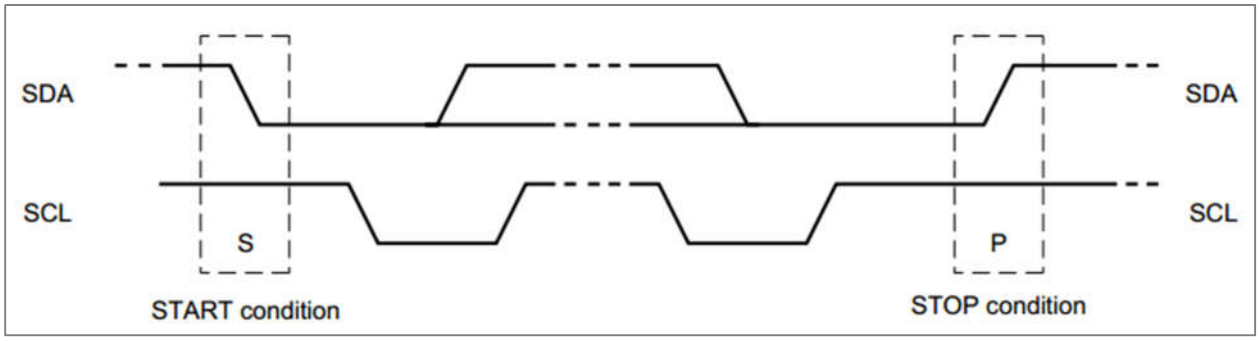

Figure 2.3 Start and Stop Condition

As you can see in Figure 2.3, a Start is issued by bringing the SDA line low while the SCL line is high. After that the Master controls the SCL line and can generate clock signals. A Stop condition is implemented by transitioning the SDA line high while the SCL line is high.

2.3 Starting Bytes

The I^2C bus is a byte-oriented protocol. After signaling Slaves by the Start condition, the Master sends "starting bytes" to the Slave. There are two components that make us the "starting bytes": Slave address and data direction (Read or Write).

The Master sends the MSB (Most Significant Bit) first and the LSB (Least Significant Bit) last. There are two addressing modes in the I^2C protocol: the 7-bit and 10-bit address modes.

We will first consider the 7-bit addressing mode. For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by R/W bit indicating whether this is a read (1) or write (0) operation.

The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address). Once the first 8 bits of the frame are sent, the receiving device is given control over SDA. If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that

the receiving device either did not receive the data or did not know how to parse the message. In that case, the exchange halts, and it's up to the master of the system to decide how to proceed.

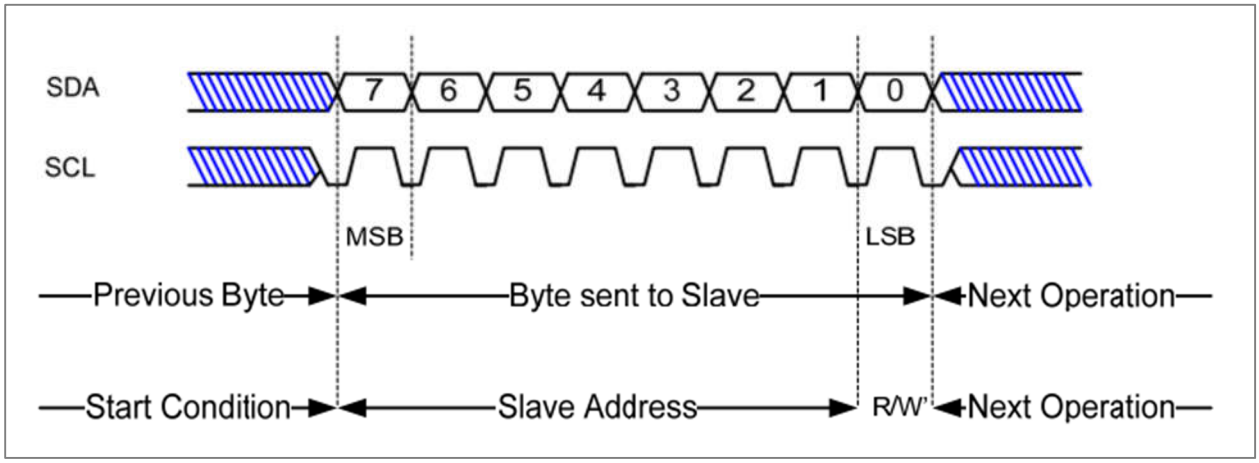

Figure 2.4 Starting Bytes

We now consider the 10-bit addressing mode. In a 10-bit addressing system, two frames are required to transmit the slave address. The first frame will consist of the code b11110xyz, where 'x' is the MSB of the slave address, y is bit 8 of the slave address, and z is the read/write bit as described above. The first frame's ACK bit will be asserted by all slaves which match the first two bits of the address. As with a normal 7-bit transfer, another transfer begins immediately, and this transfer contains bits 7:0 of the address. At this point, the addressed slave should respond with an ACK bit. If it doesn't, the failure mode is the same as a 7-bit system.

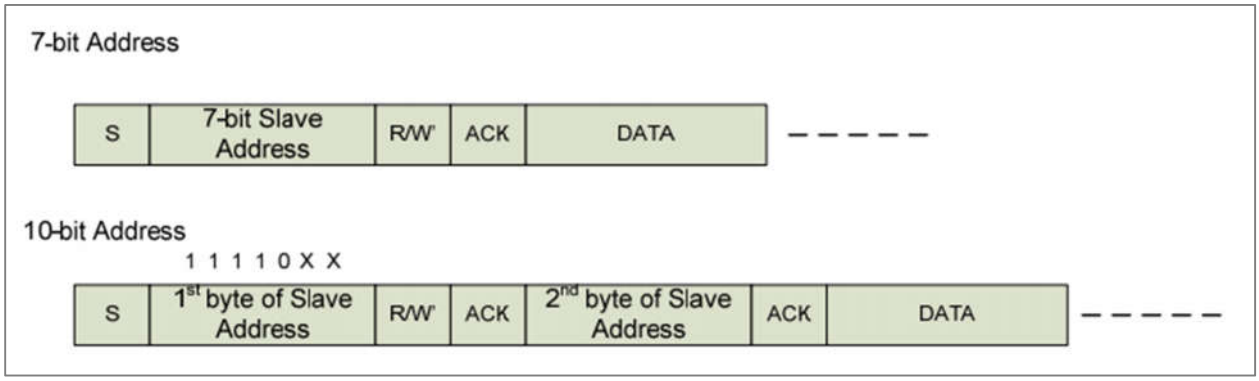

Figure 2.5 Addressing Modes

2.4 Acknowledgement

Acknowledgement is obligatory in order to inform the transmitter that data has been successfully transmitted. Figure 2.6 illustrates the acknowledgement mechanism. The Master generates the acknowledge-related clock pulse and the transmitter releases the SDA line (HIGH) during the acknowledge clock pulse so that the receiver can take control of the SDA line. If the receiver does not acknowledge, leaving the SDA line high, the transfer must be aborted. If it acknowledges by pulling the SDA line low, the transmitter knows that data has been successfully received, so it keeps sending data to the receiver.

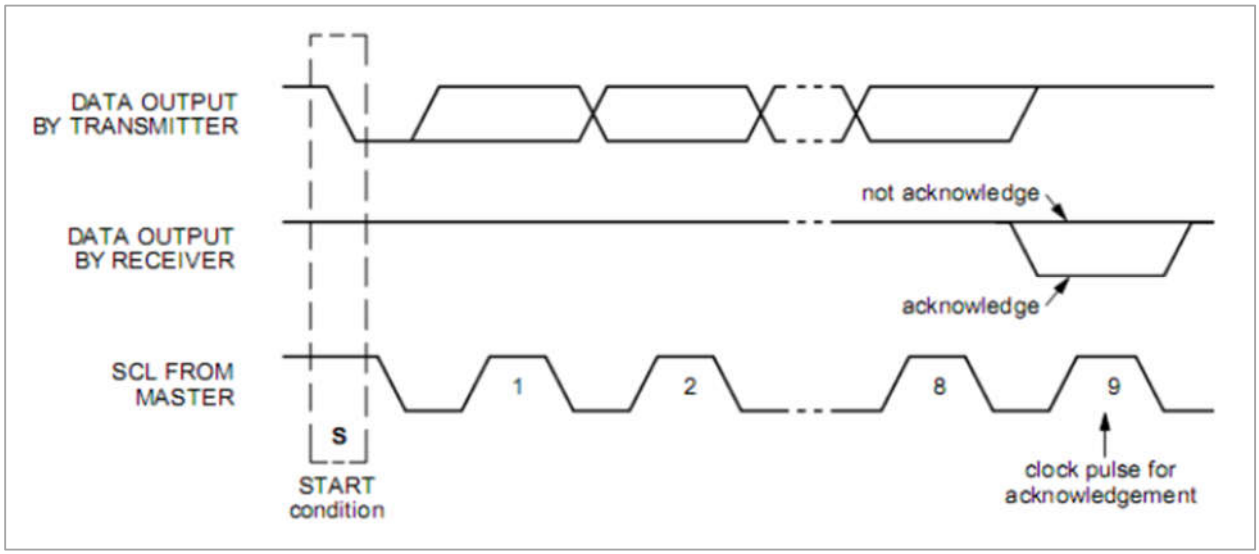

Figure 2.6 Acknowledgement Mechanism

2.5 Complete Data Transfer

All of the major aspects of I^2C bus discussed so far are combined to create a complete data transfer from the transmitter to the receiver. The 7 bit addressing mode of complete data transfer is shown in figure 2.7. SCL signal, Start and Stop signals, and the first byte must be generated by the Master. The Acknowledgement of the first byte must be generated by the Slave when it recognizes its address on the bus. The other Acknowledgements are generated by the receiver.

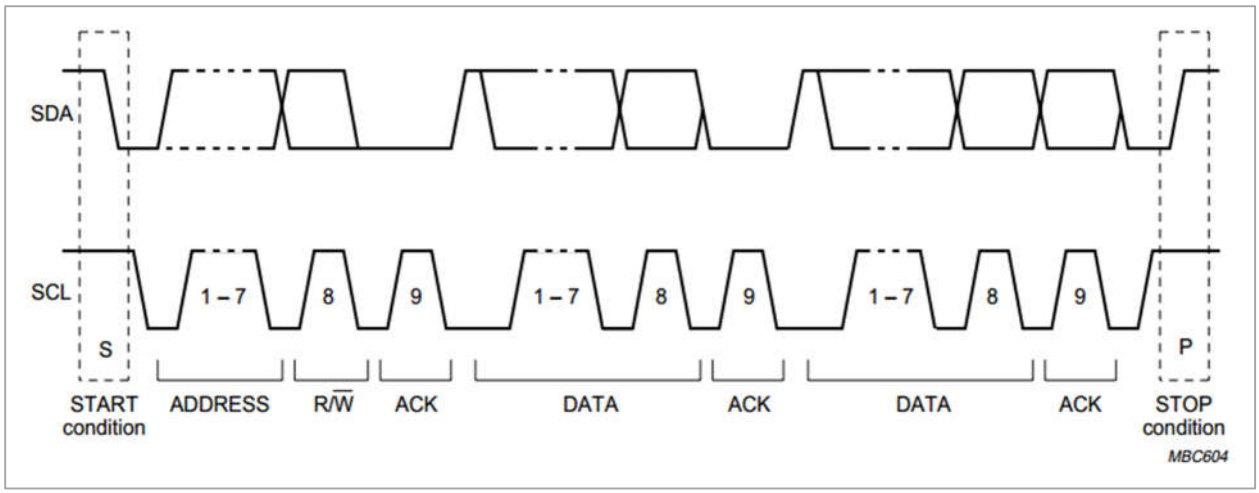

Figure 2.7 I2C Complete Data Transfer

3 I2C Slave Verilog Design

This section describes the detailed I2C Slave Verilog Design with block diagram and description of each block.

3.1 Modules

I2C Slave module is designed with modular design such that each module is assigned the dedicated task. Modular design helps in implementation and debugging.

I2C Slave Verilog design is divided into following modules:

- Slave Controller
- Start-Stop Detector
- Shifter
- Counter
- Registers
    - Slave Address Register
    - Receive Data Register
    - Transmit Data Register

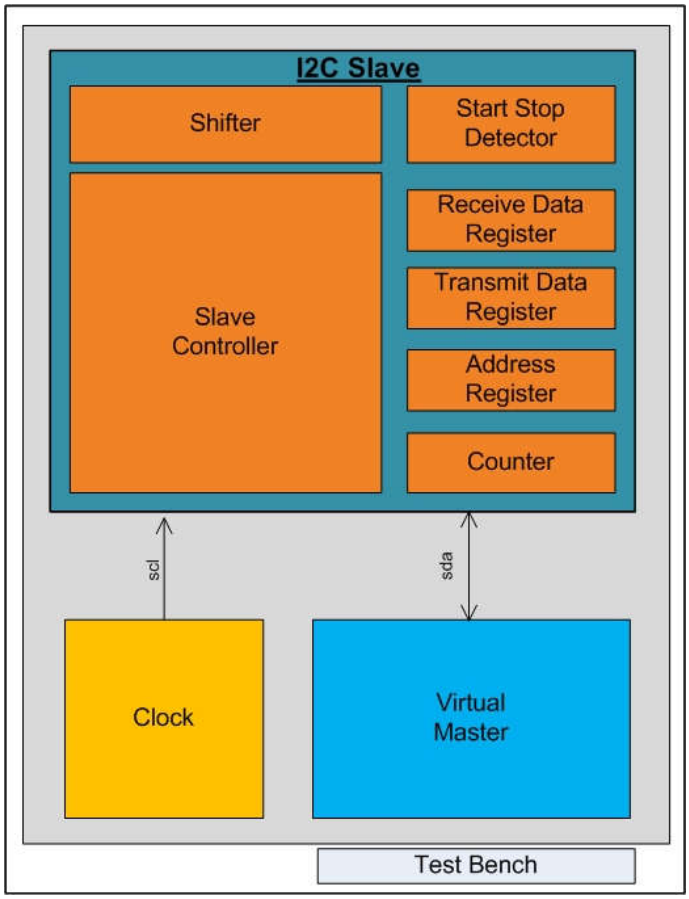

Figure 3.1 I2C Verilog Modules

To test the I2C Slave design, virtual master and clock generator is designed to provide the clock on SCL and data on SDA line.

All the modules are integrated under the testbench module. Figure 3.1 represents the pictorial view of the I2C Verilog modules.

3.2 System Block Diagram

Below figure represents the system block diagram and interconnections of the different modules with all the signals & data transfer. Each blocks are described below in detail.

3.2.1 Tri State Buffer

SCL and SDA pins of the I2C is bi-directional pins. Therefore we need to separate out the pins for providing the output on the pin and to take the input from the pin. To solve this purpose, we have used the tri state buffer which will provide the output and input pin with output enable pin.

3.2.2 Slave Controller

It is the main module of I2C Slave. It controls the other modules of the I2C slave i.e. Counter, Shifter and Start/Stop Detector. In Verilog, it is designed using Finite State Machine. Slave controller works on the Pos edge, Neg edge and also on various signals to maintain time synchronization. Finite State Machine is explained below in detail.

3.2.3 Finite State Machine

Finite State Machine is used to handle the different states in the I2C Protocol.

STATE_IDLE

This is the Initial state and whenever there is no any activity on the bus, the slave controller remains in this state. After completing the whole sequence of the I2C protocol, slave controller comes back in this initial state.

SLAVE_ADDRESS

- In this state, Slave controller enables the Counter and Shifter to store the incoming bits in the shift register.
- Shifter will continue to get the data and whenever the 8 bits are received, counter will send the signal to the slave controller indicating the completion of the 8 bits.
- After that, Slave controller disables the Shifter and gets the Slave Address.

ADDR_ACK

- Slave Address is compared with the Slave Address Register and if the address matches then it sends the ACK to the Master otherwise sends the Neg ACK.
- Slave Controller checks the R/W bit in the received data and decide whether it should go to the Slave Read mode or Slave Write mode.

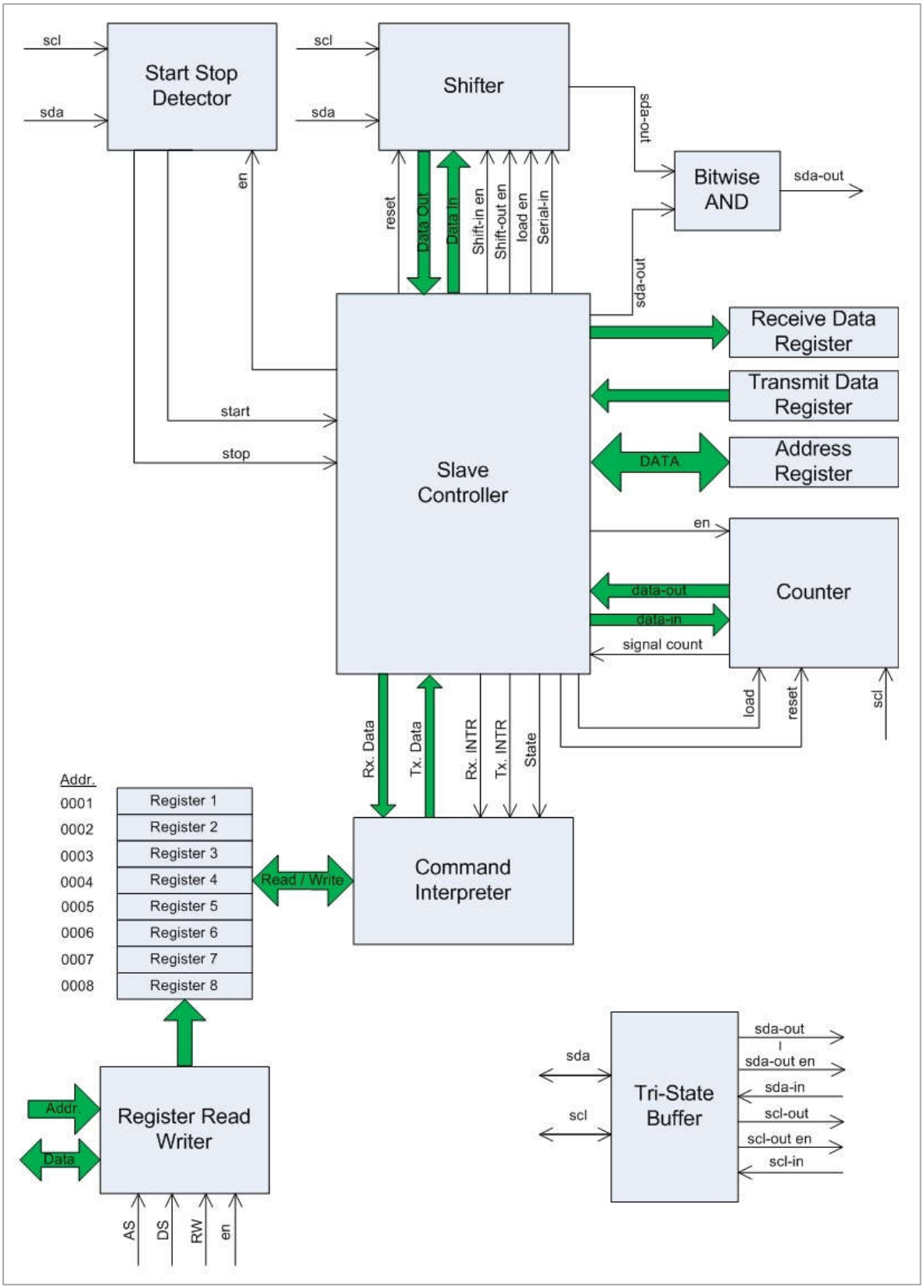

Figure 3.2 System Block Diagram

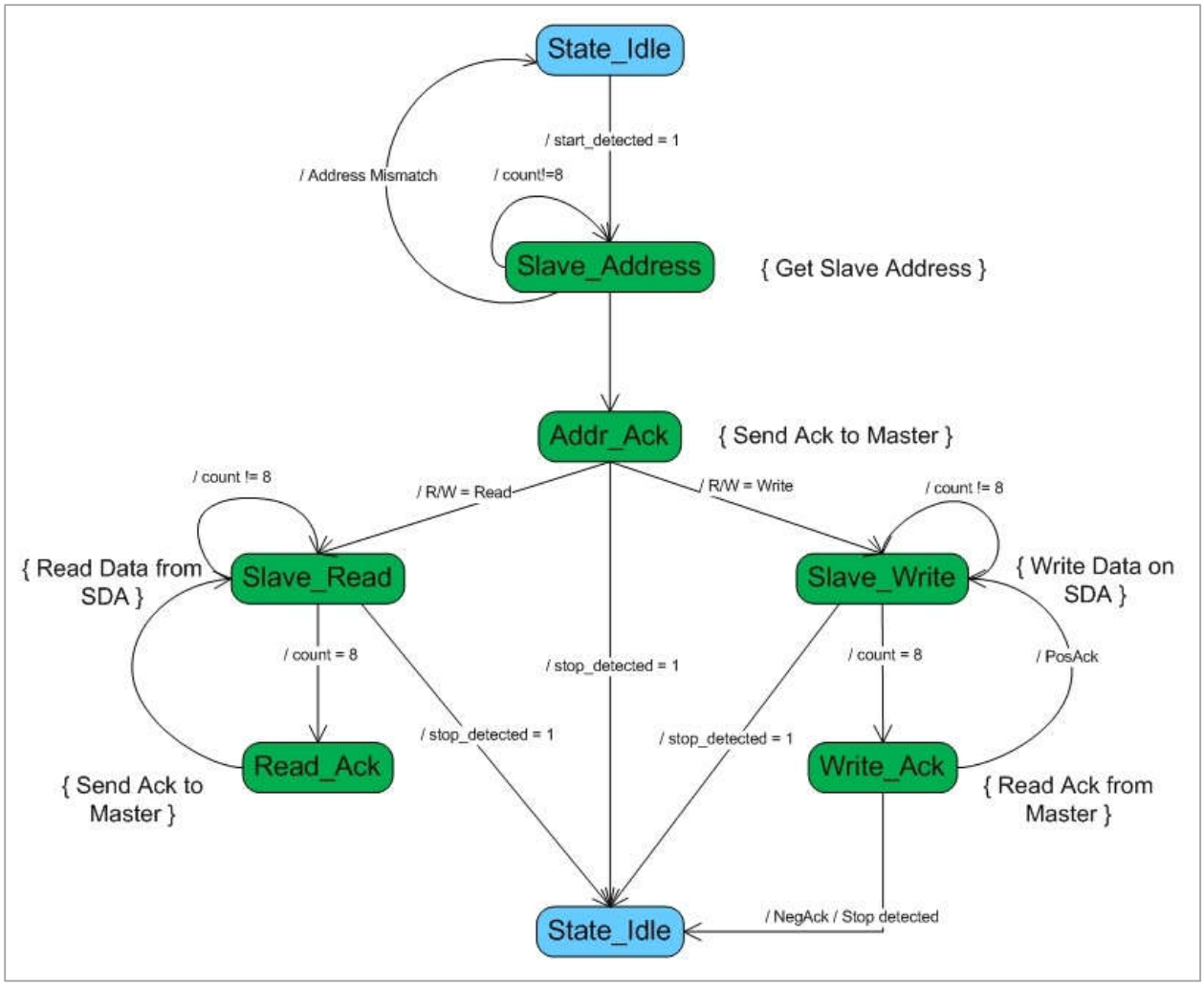

Figure 3.3 Finite State Machine

SLAVE_READ

- In this state, Slave Controller enables the Shifter and Counter to receive the 8 incoming data.
- Once the 8 bits are received, it goes to the READ_ACK mode.

READ_ACK

- Slave controller sends the positive ACK to the Master by keeping the SDA line low during the High on SCL.
- Slave controller again moves to the SLAVE_READ state and start reading the data.

SLAVE_WRITE

- Slave Controller take the data from the Tx Data Register and stores in the shifter.
- After that, Slave Controller enables the Shifter and Counter to send the stored 8 bits data.
- Once the 8 bits are sent, it goes to the WRITE_ACK mode.

WRITE_ACK

- Slave controller waits for the ACK from the Master.
- If the ACK is positive then it goes to the SLAVE_WRITE mode otherwise it goes to the STATE_IDLE mode.

3.2.4 Start Stop Detector

This module is the starting and ending point of the I2C operation. It is used to detect the start and stop conditions generated by the master. The start and stop conditions are detected on the High Pulse on the SCL line.

- START condition
    - This condition is detected when there is a transition on SDA line from high to low when SCL line is kept high.
- STOP condition
    - This condition is detected when there is a transition on SDA line from low to high when SCL line is kept high.
- It uses a combinational loop to detect the transition on the SDA line.

3.2.5 Shifter

- I2C is a serial data transfer protocol, the address and data both should be send or received serially.
- To store the address or data received or sent serially on the SDA line, every single bit received should be shifted into a register or sent from the shifter which is done using the Shifter module.
- There are various signal between Slave controller and Shifter such as DATA_OUT, DATA_IN, shift_en, load_en, serial_in and reset.

3.2.6 Counter

- Once the address or data is being sent on SDA line by either Master or Slave it must be counted.
- In the project we had set a 7-bit address which when received serially should be counted and after receiving 7-bits address a bit indicating RD/WR signal is send after that transfer should be stopped and the address is compared, this function of counting and generating the control signals Counter is used.
- Whenever the 8 bits are transferred, Counter sends the signal to the slave controller indicating the 8 bits are done.

3.2.7 Slave Address Register

- The predefined address of Slave with whom the master wants to communicate is saved in the Address Register.
- When Slave receives the address on the SDA line sent by the master, it is compared with the address which is saved in the Address Register.

3.2.8 Receive Data Register

- The data being received by the Slave during read operation should be saved somewhere. For this purpose received data register is used.

3.2.9 Transmit Data Register

- The data which should be transmitted by the Slave on the SDA line during write operation should be saved in the slave controller.
- This register holds the data and it is read by the slave controller whenever slave wants to send the data.

3.2.10 Command Interpreter & Register Read Write

- Whenever the I2C Slave module is integrated with the Sensor Module, it needs to interface with the I2C Slave module.
- Sensor module needs to implement the command interpreter which is used to read or write the different registers.

4 Simulation

4.1 Waveforms

Figure 4.1 Simulation Result

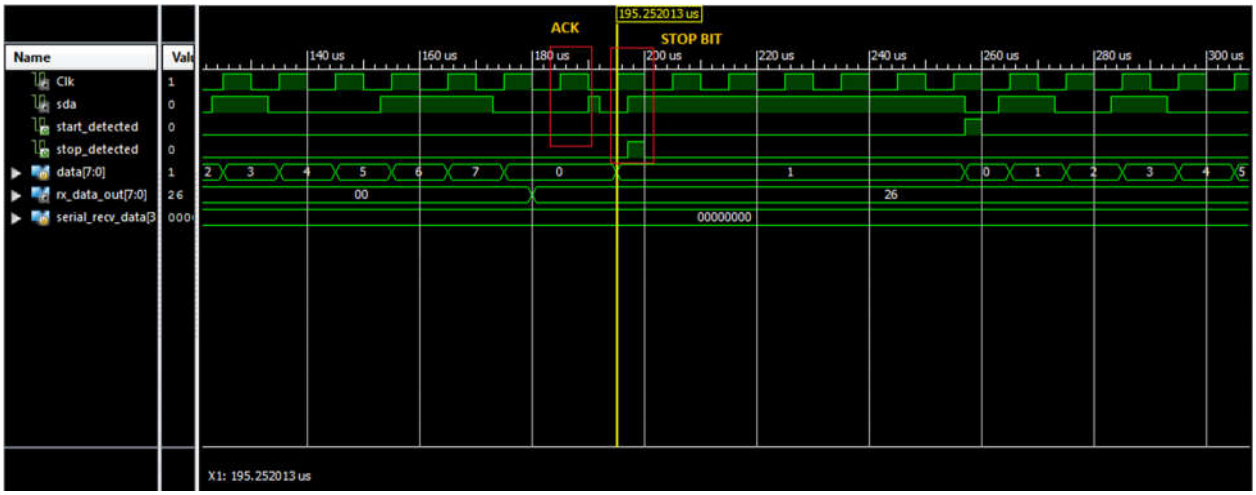

Figure 4.2 Simulation Result2

4.2 Waveform Explanation

A clock signal with frequency 100 MHz is generated.
In the figure 4.1 and 4.2 six functions are showed
START

- At 5us clk is driven high, it remains high till 10us.
- To generate a start signal SDA is driven low when clk is high, which is done at 75us.
- As soon as the transition is detected slave controller sets the slave_detected signal, indicating that address transfer should start.

SLAVE ADDRESS

- Slave address consists of 7 bit of slave address along with 1 bit of RD/WR signal.
- After detecting the start condition sending slave address is started.
- Address is detected only on the rising edge of the clock, hence the address bit which needed to be send is send on SDA line when SCL line is low and then kept stable for the full high transition of SCL line.
- In this way for the next 7 clk signals 7 address bits are send.
- The 8^{th} bit is the RD/WR bit which indicates that the slave should read or write. In the above figure a read signal is send, hence SDA line is kept low.
- The slave address in the above example is 0x53.

ACK

- After sending the address by the master, it is compared with the predefined slave address. If the match is found then ACK bit is set. If the match is not found a NEGACK is send. In the above case ACK bit is send indicating that the received address is matched**.**

DATA FROM MASTER

- After ACK bit is received 8 bit data is send on the SDA line by the Master or Slave depending on the RD/WR bit, in the same way as address is send. After 8 bits are received again an Acknowledge bit is send which indicates that 8-bit data is received.

STOP

- After receiving the ACK if the master want to send another data it is directly send, if not then master should initiate to stop the communication by sending a STOP bit as shown in the second figure.
- Same as START signal STOP signal is send. STOP signal is detected when there is a transition on SDA line from low to high.

5 FPGA Implementation

5.1 Architectural Overview

5.1.1 Block RAM

For applications requiring large, on-chip memories, Spartan® generation FPGAs provide plentiful, efficient Select RAM memory blocks. Using various configuration options, Select RAM blocks create RAM, ROM, FIFOs, large look-up tables, data width converters, circular buffers, and shift registers, each supporting various data widths and depths.

Each block RAM contains 18,432 bits of fast static RAM, 16K bits of which is allocated to data storage and, in some memory configurations, an additional 2K bits allocated to parity or additional "plus" data bits. Each block memory supports multiple configurations or aspect ratios. Cascade multiple block RAMs to create deeper and wider memory organizations with a minimal timing penalty incurred through specialized routing resources. The output register enables full-speed operation at over 250 MHz for all data widths.

5.1.2 Configurable Logic Blocks (CLBs)

The Configurable Logic Blocks (CLBs) constitute the main logic resource for implementing synchronous as well as combinatorial circuits. Each CLB contains four slices, and each slice contains two Look-Up Tables (LUTs) to implement logic and two dedicated storage elements that can be used as flip-flops or latches. The LUTs can be used as a 16x1 memory (RAM16) or as a 16-bit shift register (SRL16), and additional multiplexers and carry logic simplify wide logic and arithmetic functions. Most general - purpose logic in a design is automatically mapped to the slice resources in the CLBs. The details of the CLB resources are helpful when estimating the number of resources required for an application or when optimizing a design to the architecture.

5.1.3 CLB Array

The CLBs are arranged in a regular array of rows and columns as shown below.

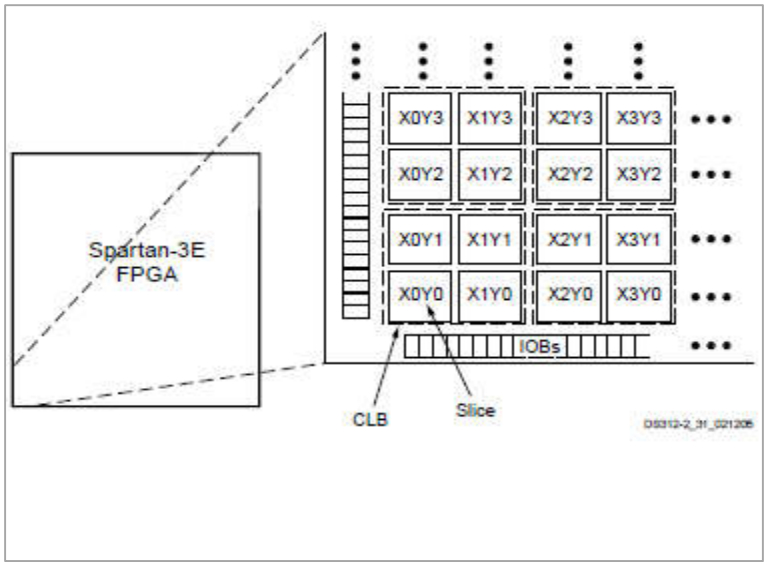

Figure 5.1 Configuration Logic Block Array

Look-Up Tables as Distributed RAM

Each Spartan® generation Configurable Logic Block (CLB) contains up to 64 bits of single-port RAM or 32 bits of dual-port RAM. This RAM is distributed throughout the FPGA and is commonly called “distributed RAM” to distinguish it from the 18-Kbit block RAM. Distributed RAM is also referred to as LUT RAM.

5.2 The CORE Generator System

The CORE Generator provides centralized access to a catalogue of ready-made IP functions ranging in complexity from simple arithmetic operators, such as adders, accumulators, and multipliers to system-level building blocks, such as filters, transforms, and memories. The CORE Generator user interface makes it very easy to access the latest Spartan generation IP releases and to get helpful, up-to-date information. The use of CORE Generator IP cores in Spartan-3 generation designs enables designers to shorten design time, and it also helps them realize high levels of performance and area efficiency without any special knowledge of the Spartan-3 generation architecture.

5.2.1 5.3.1 Boundary-Scan

Boundary-Scan testing is used to identify faulty board-level connections, such as unconnected or shorted pins. Boundary-Scan tests allow designers to quickly identify manufacturing or layout problems, which otherwise could be nearly impossible to isolate, especially with high-count ball-grid packages.

5.2.2 IEEE Standards

Joint Test Action Group (JTAG) is the commonly used name for IEEE standard 1149.1 which defines a method for Boundary-Scan. JTAG compliant devices have dedicated hardware that comprises a state machine and several registers to allow Boundary-Scan operations. This dedicated

hardware interprets instructions and data provided by four dedicated signals: TDI (Test Data In), TDO (Test Data Out), TMS (Test Mode Select), and TCK (Test Clock). The JTAG hardware interprets instructions and data on the TDI and TMS signals, and drives data out on the TDO signal. The TCK signal is used to clock the process.

5.2.3 Using I/O Resources

All signals entering and exiting a Spartan® generation FPGA must pass through the I/O resources, known as I/O blocks or IOBs. Because FPGAs are used in more advanced applications, they must support an increasing variety of I/O features. The Spartan generation FPGAs simplify high-performance design by offering selectable I/O standards for inputs and outputs. Over 20 different standards are supported in each family, with different specifications for current, voltage, I/O buffering, and termination techniques. As a result, the Spartan-3 generation FPGA can be used to integrate discrete translators and directly drive the most advance backplanes, buses, and memories. Directly providing the necessary interface standard not only eliminates the cost of external translators, but also significantly improves the chip-to-chip speed and reduces power consumption.

5.2.4 Clock Sources

The Spartan board supports three primary clock input sources:

- The board includes an on-board 50 MHz clock oscillator.
- Clocks can be supplied off-board via an SMA-style connector. The FPGA can generate clock signals or other high-speed signals on the SMA-style connector.
- Optionally install a separate 8-pin DIP-style clock oscillator in the supplied socket.

5.2.5 On-Board 50 MHz Oscillator: CLK_50MHz: (C9)

The board includes a 50 MHz oscillator with a 40% to 60% output duty cycle. The oscillators accurate to ±2500 Hz or ±50 ppm.

5.2.6 Voltage Control

The clock resources are also controlled by jumper JP9. By default, JP9 is set for 3.3V. The on-board oscillator is a 3.3V device and might not perform as expected when jumper JP9 is set for 2.5V.

5.3 CHIPSCOPE PRO TOOL:

The Xilinx Chip Scope tools package has several modules that you can add to your Verilog design to capture input and output directly from the FPGA hardware.

- ICON (Integrated Controller): A controller module that provides communication between the Chip Scope host PC and Chip Scope modules in the design (such as VIO and ILA).
- VIO (Virtual Input /Output): A module that can monitor and drive signals in design in real-time. These are virtual push-buttons (for input) and LEDs (for output). These can be used for debugging purposes, or they can be incorporated into the design as a permanent I/O interface.

- ILA (Integrated Logic Analyzer): A module that lets you view and trigger on signals in your hardware design.

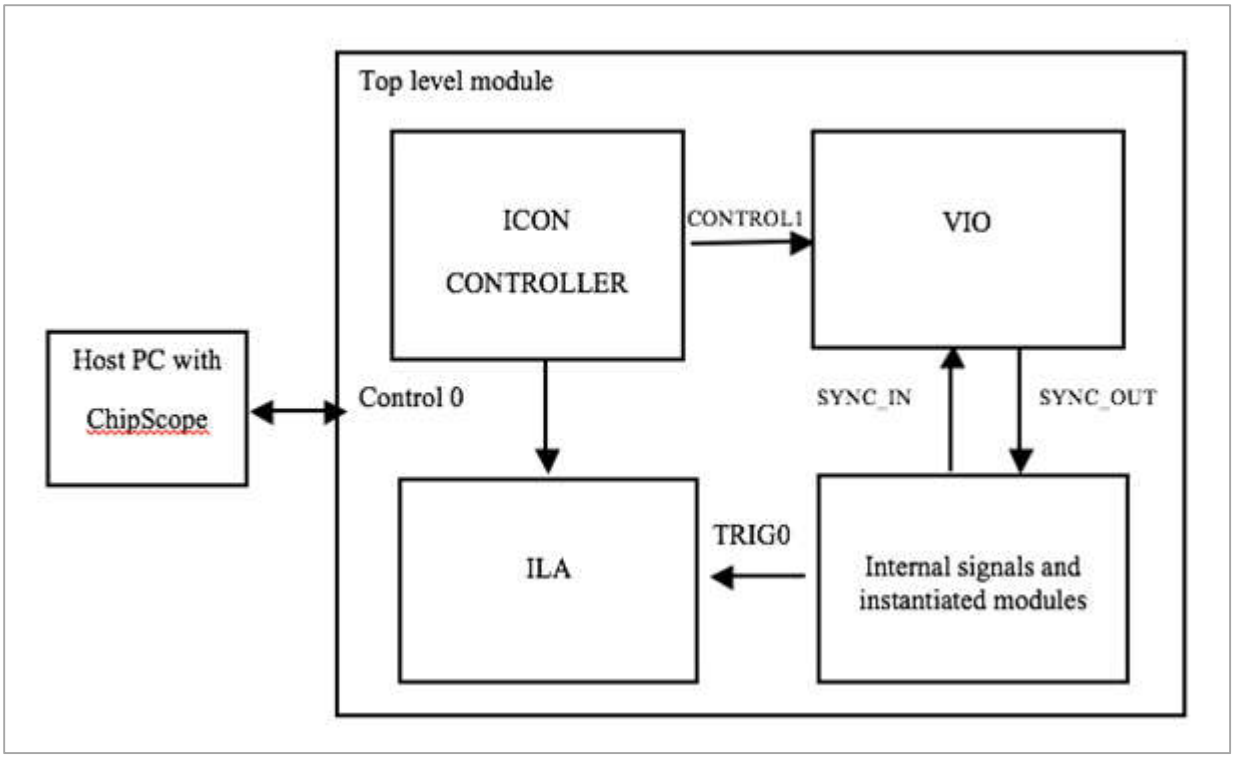

Figure 5.2 Chip Scope Pro Organization detailed diagrammatic representation

The ICON controller module communicates with the host PC and sends commands to other Chip Scope modules via a control port. Your ICON controller module must be generated with the same number of control ports as there are other Chip Scope modules in your design. For example, if you want to add an ILA module and a VIO module to your design, generation ICON module with two control ports.

Once you've added the ICON module to your design, you can add as many Chip Scope modules as you have control ports. VIO and ILA modules take the ICON control port as an input and then interact with the modules in your design through SYNC_IN/SYNC_OUT and trigger ports respectively.

Following are the Chip Scope Pro Tool Cores which are included in the modules given in detail

ICON

The Chip Scope™ Pro Integrated Controller core (ICON) provides an interface between the JTAG Boundary Scan (BSCAN) interface of the FPGA device and the Chip Scope Pro cores, including the following types of cores:

- Integrated Logic Analyzer (ILA)
- Virtual Input/Output (VIO)

- Agilent Trace Core 2 (ATC2)
- Integrated Bus Analyzer (IBA)

Features

- Provides a communication path, using the JTAG port, between the Chip Scope Pro Analyzer software and the ILA, VIO, ATC2, and IBA cores.
- Connects to the JTAG chain through the USER scan chain feature of the BSCAN component.
- Supports up to 15 connections to ILA, VIO, ATC2, and IBA cores.

ILA

The customizable Integrated Logic Analyzer (ILA) IP core is a logic analyzer that can be used to monitor the internal signals of a design. The ILA core includes many advanced features of modern logic analyzers, including Boolean trigger equations, and edge transition triggers. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components of the ILA core.

Features

- User-selectable trigger width, data width, and data depth
- Multiple probe ports, which can be combined into a single trigger condition

VIO CORE

The LogiCORE™ IP Chip Scope™ Pro Virtual Input/Output (VIO) core is a customizable core that can both monitor and drive internal FPGA signals in real time. Two different kinds of inputs and two different kinds of outputs are available, both of which are customizable in size to interface with the FPGA design.

Features

- Provides virtual LEDs and other status indicators through asynchronous and synchronous input ports
- Has activity detectors on input ports to detect rising and falling transitions between samples
- Provides virtual buttons and other controls through asynchronous and synchronous output ports
- For synchronous outputs, provides ability to define a pulse train, which is a 16-cycle train of ones and zeros that run at design speed.

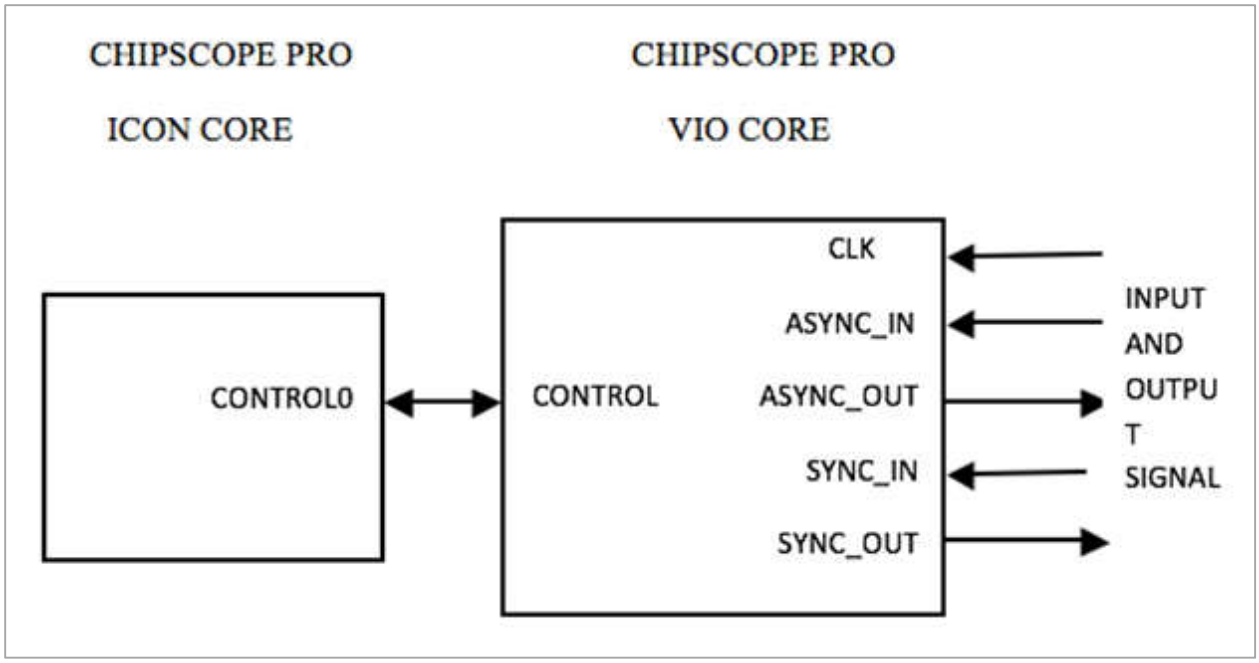

Figure 5.3 ILA CORE connection to ICON CORE

Synthesizing, Implementing, and Running Your Design

1. Xilinx ISE Design Suite 14.2

(a) Choose where to save project files.

- Select Top Level Module icon and right click on it and select New Source option.
- In New Source, select IP (Core Generator and Architecture Wizard).
- This is where the ICON and ILA cores (and their metadata files) will be generated.

(b) Under the Part tab, select all of the same device settings as you do when you setup a normal Xilinx ISE 14.2 Project.

(c) Under the Generation tab, change Design Entry to Verilog.

2. On the main screen, go to the View by Function tab.

(a) Select the Debug and Verification folder.

(b) Select Chip Scope Pro.

(c) You should now be looking at a list of Chip Scope "cores." Among them should be ICON, ILA, and VIO. We will be using ICONs and ILAs.

A: Generating the ICON

Double-click on ICON (Chip Scope Pro Integrated Controller). A window that will allow you to customize your ICON should have appeared.

(a) Component Name: Assign your ICON a name (this is arbitrary).

(b) Select the correct Number of Control Ports.

- Each ILA requires one control port.
- Normally you will only need 1 Control Port.

The Chip Scope Pro Core Generator will now generated the ICON core according to the settings specified in the directory.

Generating the ILA

- Once again on the main screen, double-click on ILA (Chip Scope Pro - Integrated Logic Analyzer). A window that will allow you to customize your ILA should have appeared.
- Component Name: Same as with the ICON
- Normally you will only need 2 Trigger Port.
- Data Port Width is only available when using separate trigger and data ports. Set this to the number of bits to see in the wave window of Chip Scope.
- Set the Trigger Port Width to the number of bits that need to trigger on.

The Chip Scope Pro Core Generator will now generated the ICON core.

Generating the VIO

- Double click on VIO core. By selecting VIO core generator, a window will allow you to customize your VIO.
- Select the SYNC IN option which interface the output of the program to be treated as input to VIO core for displaying it on Chip Scope Pro as Virtual LEDs.
- Select the SYNC OUT option which interface the input of the program to be treated as output to VIO core for displaying it on Chip Scope Pro as Virtual Switches.

Generated VIO core will be displayed in the Top Module.

Connecting the Cores to the Design.

Declare a control bus for each ILA, similar to wire [35:0] ILAControl;

- Each ILA will require a control bus.
- Route control busses as input/outputs to instantiate the ICON
- Instantiate the ICON core.
- Instantiate the ILA core.

Instantiate ILAs wherever they are necessary, just make sure to route the control bus from the ICON appropriately

ILA may have different ports.

Generate programming file and manage configurations using iMPACT, a tool featuring batch and GUI operations, allows you to perform Device Configuration and File Generation.

Analyze design and generated program inside FPGA using Chip Scope Pro Logic Analyzer Tool and observe the waveforms of the internal signals on ILA window and giving inputs and observing output using Virtual Input Switches and Virtual Output LEDs.

6 Accomplishment

The main Goal of the project was to design a ready to use I2C Slave Module Design for Sensors using Verilog HDL simulator by designing various modules used in I2c protocol which was successfully accomplished.

This I2C module has many advantages in various industries like:

- Most of the Sensors supports I2C Interface for communication
- Design new chips with easy interface
- Less time for Design and Development of new chips
- Different modules now can be connected using a single I2C module.

Furthermore while completing this project we are also able to gain and enhance our knowledge on the following topics.

- I2C Protocol Understanding
- Xilinx ISE Design Suite Learning
- Verilog Language Learning
- Project Development Life Cycle
- Debugging Skills & Bug Fixing.

7 Limitation & Future Enhancements

The I2C Slave module is designed using the Verilog HDL targeting the Sensor vendors. As the design is focused for the basic implementation, there is always scope for the future enhancements.

Limitations

- The I2C design supports only 100 KHz frequency mode of the I2C
- The design supports to receive and transmit the data into the register. To implement the Protocol for the internal register read and write, other module needs to be implemented and integrated with this module.
- The design is burned on the FPGA but could not test the signals at various points using the Chip Scope due to the unavailability of the chip scope license.

Future Enhancements

- Extend the design to support 400KHz and High Speed Mode
- Provide the support for internal register read write by implementing the required modules
- Test the design using the licensed Chip scope on the various points in the I2C Slave

8 References

- Philips I2C Specification
- Verilog Coding
  http://www.asic-world.com/verilog/index.html
- Clocking using Chip scope pro
  http://www.xilinx.com/support/documentation/user_guides/ug382.pdf
- Design of I2C Interface for Custom ASICS Used in the Detection of Ionizing Radiation by Nam Nguyen
- Xilinx Software and Documentation
  http://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html
- Chip scope pro software and cores guide
  http://www.xilinx.com/ise/verification/chipscope_pro_sw_cores_10_1_ug029.pdf